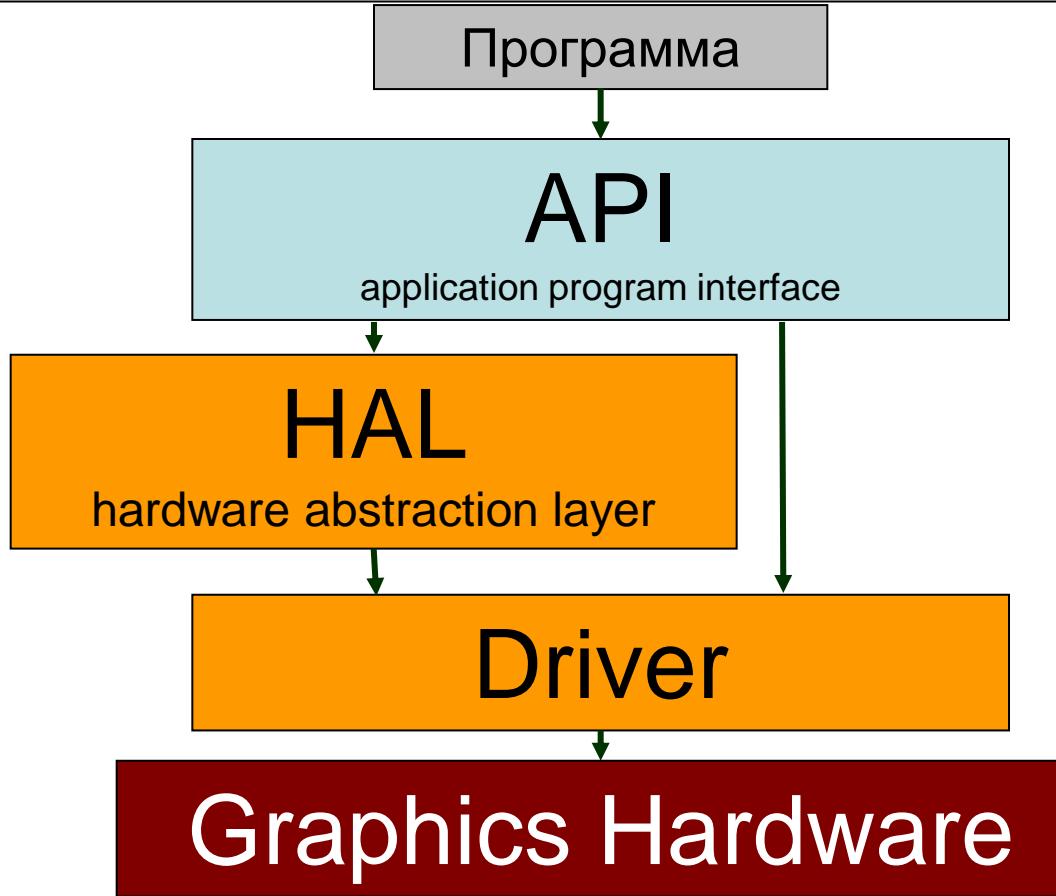
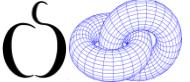


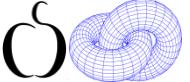
# Графика реального времени. OpenGL.

материалы занятий: <https://compsciclus.ru/courses/graphics2018/2018-autumn/classes/>  
дублируются на сайте: <http://www.school130.spb.ru/cgsg/cgc2018/>



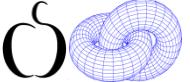


- **OpenGL** – open graphics library  
(SGI, 90-е годы, версии 1.0-4.2)
  - GLUT - OpenGL Utility Toolkit (<http://www.opengl.org/resources/libraries/glut/>)
  - FreeGLUT – Free OpenGL Utility Toolkit (<http://sourceforge.net/projects/freeglut/>)
  - GLFW - Graphics Library Framework (<http://www.glfw.org>)
  - GLEW - OpenGL Extension Wrangler Library (<http://glew.sourceforge.net/>)
  - GLM - OpenGL Mathematics (<http://glm.g-truc.net/>)
- **Microsoft DirectX Graphics** – часть MS DirectX API (Direct3D)  
(1995 Microsoft Corp., Windows Game SDK, версии 1.0-12.0)
  - D3DX – retained mode toolkit
  - XNA - Xbox New Architecture (<http://msdn.microsoft.com/ru-ru/xna/>)
  - Microsoft DXR (DirectX Raytracing)
- **Vulkan** – кроссплатформенный API для 2D и 3D графики  
(2016 Khronos Group, версии 1.0-1.195)
  - Vulkan API (<http://www.khronos.org/vulkan/>)
- **Metal** – Apple Inc. (2014, macOS, iOS,)



- Анимация (основной цикл программы)
  - Взаимодействие с ОС
  - Синхронизация по времени
  - Опрос устройств ввода
- Вывод (рендеринг)
  - Инициализация API
  - Вывод кадра
  - Хранение геометрических объектов





**Old style:** (классический вариант со стандартной библиотекой)

```
#include <time.h>
clock() -> tick -> CLOCKS_PER_SEC
```

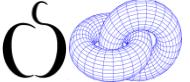
на старте программы:

```
clock_t StartTime = clock();
. . .
```

на каждом шаге:

```
clock_t t = clock();
GlobalTimeInSec = (t - StartTime) / (double)CLOCKS_PER_SEC;
```





High resolution timer: (*вариант с таймеров высокого разрешения - WinAPI*)

```
unsigned long long
StartTime, /* Start program time */
TimePerSec; /* Timer resolution */
```

на старте программы:

```
LARGE_INTEGER t;
```

определить точность таймера (тики в секунду)

```
QueryPerformanceFrequency(&t);
TimePerSec = t.QuadPart;
```

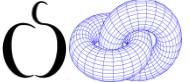
определить время начала

```
QueryPerformanceCounter(&t);
StartTime = t.QuadPart;
```

на каждом шаге:

```
LARGE_INTEGER t;
QueryPerformanceCounter(&t);
GlobalTimeInSec = (double)(t.QuadPart - StartTime) / TimePerSec;
```





-- время начала программы

StartTime

-- общее время работы - *global time*

t - StartTime

-- межкадровая задержка времени (без учета паузы) - *global delta time*

t - OldTime (*OldTime* - время прошлого кадра)

-- pause flag:

IsPause

-- локальное время (с учетом паузы) - *Local time*

t - PauseTime - StartTime

-- межкадровая задержка с учетом паузы - *local delta time*

== global delta time если !IsPause

== 0 если IsPause

для вычисления PauseTime:

если IsPause:

PauseTime += t - OldTime

-- для определения количества кадров в секунду - *frames per second (FPS)*:

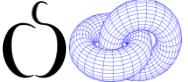
1 time per second:

FrameCounter - счетчик кадров

OldTimeFPS

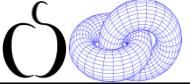
- время прошлого замера FPS





```
LARGE_INTEGER t;  
QueryPerformanceCounter(&t);  
.  
.*  
/* FPS */  
FrameCounter++;  
if (t.QuadPart - OldTimeFPS > TimePerSec)  
{  
    FPS = FrameCounter * TimePerSec / (DBL)(t.QuadPart - OldTimeFPS);  
    OldTimeFPS = t.QuadPart;  
    FrameCounter = 0;  
}
```





Файлы заголовков:

```
#include <gl/gl.h>
#include <gl/glu.h>
```

Библиотеки:

`opengl32 + glu32`

Об именах в GL:

Сокращения: **GL** – OpenGL      **GLU** — вспомогательные функции

**GLUT** — интерфейсная часть и т.п. от GLUT

**GLEW** — библиотека работы с расширениями

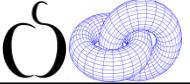
префиксы:

**glNameName** — функции: `glClear(0)` ; `glPolygonMode(...)` ;

**GL\_NAME\_NAME** — константы: `GL_FRONT_AND_BACK`, `GL_POINTS`

**Glname** — внутренние типы: `GLbyte`

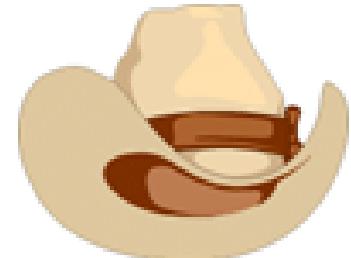




## OpenGL Extension Wrangler Library

Подключение расширений – дополнений и новых функций (необходим всегда для современных версий, подключает все новые возможности OpenGL, беря на себя нужные инициализации)

<http://glew.sourceforge.net>



### Подключение к проекту:

```
#define GLEW_STATIC      <-- отказ от glew32.dll  
#include <glew.h>       <-- подключается до GL/***.H  
+ библиотека: glew32s
```

# Graphics Library Utility Toolkit

**GLUT** (~1997)

[http://www.opengl.org/resources/libraries/glut/glut\\_downloads.php](http://www.opengl.org/resources/libraries/glut/glut_downloads.php)

**FreeGLUT** (~2011)

<http://sourceforge.net/projects/freeglut/>

*Обработка всего, что связано с интерфейсом: окна, создание "цикла" сообщений и т.п., организация ввода (мышь, клавиатура, джойстик и т.д.).*

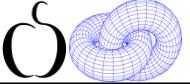
*События обрабатываются с помощью функций "обратного вызова" (CALLBACK).*

Подключение

**#include "glut.h"**

+ библиотека **glut32** (обычно подключается через **glut.h**)

В OS Windows необходима динамическая библиотека **glut32.dll**



```
#include <stdlib.h>
#include <glut.h>

void Display( void )
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    ...
    glFinish();
    glutSwapBuffers();
    glutPostRedisplay();
}

void Keyboard( unsigned char Key, int X, int Y )
{
    if (Key == 27)
        exit(0);
}
```

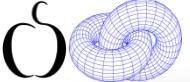
```
int main( int argc, char *argv[] )
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);

    glutInitWindowPosition(0, 0);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Sample");

    glutDisplayFunc(Display);
    glutKeyboardFunc(Keyboard);

    glutMainLoop();
    return 0;
}
```





```
#include <windows.h>

#define WND_CLASS_NAME "My window class"

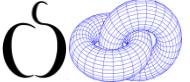
/* Главная функция программы */
INT WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
                     CHAR *CmdLine, INT ShowCmd )
{
    WNDCLASS wc;
    HWND hWnd;
    MSG msg;

    wc.style = CS_VREDRAW | CS_HREDRAW;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hbrBackground = (HBRUSH)COLOR_WINDOW;
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.lpszMenuName = NULL;
    wc.hInstance = hInstance;
    wc.lpfnWndProc = MyWindowFunc;
    wc.lpszClassName = WND_CLASS_NAME;

    /* Регистрация класса в системе */
    if (!RegisterClass(&wc))
    {
        MessageBox(NULL, "Error register window class", "ERROR", MB_OK);
        return 0;
    }
}
```

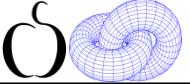
```
/* Создание окна */
hWnd =
    CreateWindow(WND_CLASS_NAME, "Title",
                 WS_OVERLAPPEDWINDOW,
                 CW_USEDEFAULT, CW_USEDEFAULT,
                 CW_USEDEFAULT, CW_USEDEFAULT,
                 NULL, NULL, hInstance, NULL);
/* Показать и перерисовать окно */
ShowWindow(hWnd, SW_SHOWNORMAL);
UpdateWindow(hWnd);
/* Цикл обработки сообщений */
while (TRUE)
{
    if (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
    {
        if (msg.message == WM_QUIT)
            break;
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    else
        . . .
}
return msg.wParam;
} /* End of 'WinMain' function */
```



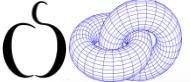


```
/* Функция обработки сообщения окна */
LRESULT CALLBACK MyWindowFunc( HWND hWnd, UINT Msg,
                               WPARAM wParam, LPARAM lParam )
{
    HDC hDC;
    PAINTSTRUCT ps;
    int w, h;

    switch (Msg)
    {
    case WM_CREATE:
        SetTimer(hWnd, 30, 3, NULL);
        return 0;
    case WM_SIZE:
        w = LOWORD(lParam);
        h = HIWORD(lParam);
        return 0;
    case WM_ERASEBKGND:
        return 1;
    case WM_TIMER:
        return 0;
    case WM_PAINT:
        hDC = BeginPaint(hWnd, &ps);
        EndPaint(hWnd, &ps);
        return 0;
    case WM_DESTROY:
        KillTimer(hWnd, 30);
        PostQuitMessage(0);
        return 0;
    }
    return DefWindowProc(hWnd, Msg, wParam, lParam);
} /* End of 'MyWindowFunc' function */
```



OS	Animation	Application	Render3D
WinMain: - регистрация класса окна - создание окна - цикл сообщений <i>while (GetMessage(...)) DispatchMessage(...);</i>	<b>AnimInit();</b> инициализация	массив элементов анимации <b>UNIT:</b> [	<b>RndInit();</b>
WinFunc: <b>WM_GETMINMAXINFO:</b> ... <b>WM_CREATE:</b> ... <b>WM_SIZE:</b> ... <b>WM_TIMER:</b> ... <b>WM_ERASEBKGND:</b> ... <b>WM_PAINT:</b> ... <b>WM_DESTROY:</b> ... Input: <b>WM_MOUSEWHEEL:</b> ... <b>WM_MOUSE***</b> <b>WM_KEY***</b> ...	<b>AnimClose();</b> денициализация	<b>Init(Anim);</b> инициализация	<b>RndClose();</b>
	<b>AnimResize(W, H);</b> изменение размера кадра	<b>Close(Anim);</b> денициализация	<b>RndResize(W, H);</b>
	<b>AnimCopyFrame();</b> копирование кадра	<b>Response(Anim);</b> отклик элемента анимации на смену кадра (обработка клавиатуры, таймера и т.п.)	<b>RndStart();</b>
	<b>AnimRender();</b> построение кадра (опрос устройств ввода, обновление таймера, опрос всех элементов анимации и вызов у них функции <b>Response</b> , очистка кадра и вызов у всех функций анимации <b>Render</b> )	<b>Render(Anim);</b> отрисовка элемента анимации	<b>RndEnd();</b>
	+КОНТЕКСТ анимации <b>ANIM Anim;</b>	]	<b>RndCopyFrame();</b>
		в каждую функцию элемента анимации приходит параметр - <b>Anim</b> - текущий контекст (параметры) анимации: - клавиатура - мышь - джойстик - таймер - параметры ввода - параметры визуализации	<b>RndPrim:</b> Create/Load/Free/Draw



# WinAPI: инициализация OpenGL

```
/* Windows specific global data */
HWND VG4_hRndWnd;
HDC VG4_hRndDC;
HGLRC VG4_hRndGLRC;

RndInit:
INT i;
PIXELFORMATDESCRIPTOR pfd = {0};

/* Store window and create device context */
VG4_hRndWnd = hWnd;
VG4_hRndDC = GetDC(hWnd);
/* OpenGL init: pixel format setup */
pfd.nSize = sizeof(PIXELFORMATDESCRIPTOR);
pfd.nVersion = 1;
pfd.dwFlags = PFD_DOUBLEBUFFER | PFD_SUPPORT_OPENGL;
pfd.cColorBits = 32;
pfd.cDepthBits = 32;
i = ChoosePixelFormat(VG4_hRndDC, &pfd);
DescribePixelFormat(VG4_hRndDC, i, sizeof(pfd), &pfd);
SetPixelFormat(VG4_hRndDC, i, &pfd);

/* OpenGL init: setup rendering context */
VG4_hRndGLRC = wglCreateContext(VG4_hRndDC);
wglMakeCurrent(VG4_hRndDC, VG4_hRndGLRC);
```

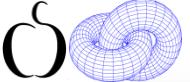
```
/* GLEW initialize */
if (glewInit() != GLEW_OK ||
    !(GLEW_ARB_vertex_shader & GLEW_ARB_fragment_shader))
{
    wglGetCurrent(NULL, NULL);
    wglDeleteContext(VG4_hRndGLRC);
    ReleaseDC(VG4_hRndWnd, VG4_hRndDC);
    exit(0);
}
/* OpenGL set render parameters */
glClearColor(0.3, 0.5, 0.7, 1);

glEnable(GL_DEPTH_TEST);
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

glEnable(GL_PRIMITIVE_RESTART);
glPrimitiveRestartIndex(-1);

RndClose:
wglGetCurrent(NULL, NULL);
wglDeleteContext(VG4_hRndGLRC);
ReleaseDC(VG4_hRndWnd, VG4_hRndDC);
```





# Обработка вывода OpenGL

```
/*** Rendering specific data ***/
/* Projection data */
float
VG4_RndProjSize = 0.1,      /* Project plane unit square size */
VG4_RndProjDist = 0.1,      /* Distance to project plane (near) */
VG4_RndProjFarClip = 1000; /* Distance to project far clip plane (far) */
int
VG4_RndFrameW, VG4_RndFrameH; /* Viewport size */
/* Transformation matrix */
MATR
VG4_RndViewMatr, /* View coordinate system matrix */
VG4_RndProjMatr; /* Projection matrix */
/* Camera parameters */
VEC
VG4_RndCamLoc, /* Camera location */
VG4_RndCamRight, /* Camera right direction */
VG4_RndCamUp, /* Camera up direction */
VG4_RndCamDir; /* Camera forward direction */

RndResize:
    float ratio_x = 1, ratio_y = 1;

    if (VG4_RndFrameW >= VG4_RndFrameH)
        ratio_x *= (float)VG4_RndFrameW / VG4_RndFrameH;
    else
        ratio_y *= (float)VG4_RndFrameH / VG4_RndFrameW;

    VG4_RndProjMatr =
        MatrFrustum(-ratio_x * VG4_RndProjSize / 2, ratio_x * VG4_RndProjSize / 2,
                    -ratio_y * VG4_RndProjSize / 2, ratio_y * VG4_RndProjSize / 2,
                    VG4_RndProjDist, VG4_RndProjFarClip);
    glViewport(0, 0, W, H);
```

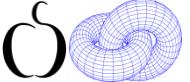
```
RndCopyFrame:
    SwapBuffers(VG4_hRndDC);

RndStart:
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

RndEnd:
    glFinish();

AnimRender:
    // опрос устройств ввода
    // опрос таймера
    // вызов у объектов анимации Response
    VG4_RndStart();
    // вызов у объектов анимации Render
    VG4_RndEnd();
```





# OpenGL: Устаревший вывод

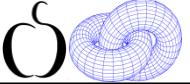
```
glBegin(режим задания примитивов);
...
команды задания примитивов
glEnd();
```

Например:

```
glBegin(GL_TRIANGLES);
    glColor3d(1, 1, 0);
    glVertex3d(0.15, 0, 0.15);
    glVertex3d(0.15, 0, 8.8);
    glVertex3d(8.8, 0, 0.15);
glEnd();
```

Например, рисование осей X, Y и Z:

```
glBegin(GL_LINES);
    glColor3d(1, 0, 0);
    glVertex3d(0, 0, 0);
    glVertex4d(1, 0, 0, 0);
    glColor3d(0, 1, 0);
    glVertex3d(0, 0, 0);
    glVertex4d(0, 1, 0, 0);
    glColor3d(0, 0, 1);
    glVertex3d(0, 0, 0);
    glVertex4d(0, 0, 1, 0);
glEnd();
```



# Примитивы OpenGL

Типы примитивов:

V<sub>4</sub>  
V<sub>3</sub>

V<sub>5</sub>  
V<sub>2</sub>

V<sub>0</sub>  
V<sub>1</sub>

GL\_POINTS

V<sub>4</sub>  
V<sub>3</sub>

V<sub>5</sub>  
V<sub>2</sub>

V<sub>0</sub>  
V<sub>1</sub>

GL\_LINES

V<sub>4</sub>  
V<sub>3</sub>

V<sub>5</sub>  
V<sub>2</sub>

V<sub>0</sub>  
V<sub>1</sub>

GL\_LINE\_STRIP

V<sub>4</sub>  
V<sub>3</sub>

V<sub>5</sub>  
V<sub>2</sub>

V<sub>0</sub>  
V<sub>1</sub>

GL\_LINE\_LOOP

V<sub>4</sub>  
V<sub>3</sub>

V<sub>5</sub>  
V<sub>2</sub>

V<sub>0</sub>  
V<sub>1</sub>

GL\_POLYGON

V<sub>4</sub>  
V<sub>3</sub>

V<sub>5</sub>  
V<sub>2</sub>

V<sub>0</sub>  
V<sub>1</sub>

GL\_TRIANGLES

V<sub>3</sub>  
V<sub>1</sub>

V<sub>5</sub>  
V<sub>2</sub>

V<sub>6</sub>  
V<sub>4</sub>

GL\_TRIANGLE\_STRIP

V<sub>5</sub>  
V<sub>4</sub>

V<sub>3</sub>  
V<sub>2</sub>

V<sub>0</sub>  
V<sub>1</sub>

GL\_TRIANGLE\_FAN

V<sub>6</sub>  
V<sub>5</sub>

V<sub>7</sub>  
V<sub>4</sub>

V<sub>3</sub>  
V<sub>0</sub>

GL\_QUADS

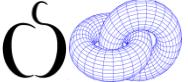
V<sub>7</sub>  
V<sub>5</sub>

V<sub>6</sub>  
V<sub>4</sub>

V<sub>2</sub>  
V<sub>0</sub>

GL\_QUAD\_STRIP





## -- Точечные примитивы

- `GL_POINTS` — отрисовка отдельных точек (каждая вершина - отдельная точка)

Дополнительные параметры точечных объектов:

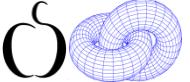
- Размер точки  
`glPointSize(размер);`
- Сглаживание для точек (круглые окончания)  
 `glEnable(GL_POINT_SMOOTH);` — включение (выключение —  `glDisable(GL_POINT_SMOOTH);`)

## -- Линейные примитивы

- `GL_LINES` — отрисовка отдельных отрезков (каждая пара вершин - отрезок)

Дополнительные параметры линейных объектов:

- Ширина линии  
`glLineWidth(ширина);`
- Сглаживание линий  
 `glEnable(GL_LINE_SMOOTH);`
- Интерполяция цвета между вершинами  
 `glShadeModel(GL_FLAT или GL_SMOOTH);` — включение (`GL_SMOOTH` — включен по умолчанию)
- Шаблон  
 `glEnable(GL_LINE_STIPPLE);` — включение отрисовки линии по шаблону задание шаблона:  
 `glLineStipple(множитель, 16-ти битный шаблон);`
- `GL_LINE_STRIP` — отрисовка ломаной
- `GL_LINE_LOOP` — отрисовка замкнутой ломаной



## -- Площадные примитивы

➤ `GL_TRIANGLES` — отрисовка треугольника (каждая тройка вершин)

Дополнительные параметры площадных объектов:

- `GL_TRIANGLE_STRIP` — полоса треугольников
- `GL_TRIANGLE_FAN` — веер треугольников
- `GL_QUADS` — каждые 4 вершины - четырехугольник (выпуклый) [устаревший]
- `GL_QUAD_STRIP` — полоса четырехугольников [устаревший]
- `GL_POLYGON` — выпуклый многоугольник [устаревший]

- У площадных фигур определено понятие лицевой и тыльной стороны. Определяется это по обходу вершин — по умолчанию обходя вершины против часовой стрелки считается, что смотрим на лицевую сторону.

`glFrontFace(GL_CW или GL_CCW);` — определяет какая сторона лицевая

(`GL_CW` — *clock-wise* — по часовой стрелке, `GL_CCW` — *counter-clock-wise* — против — по умолчанию)

- Отмена рисования определенных сторон (*culling*)

`glEnable(GL_CULL_FACE);` — включение “нерисования” определенных сторон (по умолчанию — тыльных)  
кого не рисовать:

`glCullFace(GL_BACK или GL_FRONT или GL_FRONT_AND_BACK);`

- Сглаживание (устранение ступенчатости)

`glEnable(GL_POLYGON_SMOOTH);` — сглаживание

- Интерполяция цвета (закраска по *Guro* [*Gouraud shading*])

`glShadeModel(режим интерполяции цвета — GL_FLAT без сглаживания, GL_SMOOTH — со сглаживанием);`

- Использование трафарета

`glEnable(GL_POLYGON_STIPPLE);` — отрисовка многоугольников по трафарету:

`glPolygonStipple(маска трафарета);` — задает маску — битовый массив 32 на 32 точки (`long m[32];`)

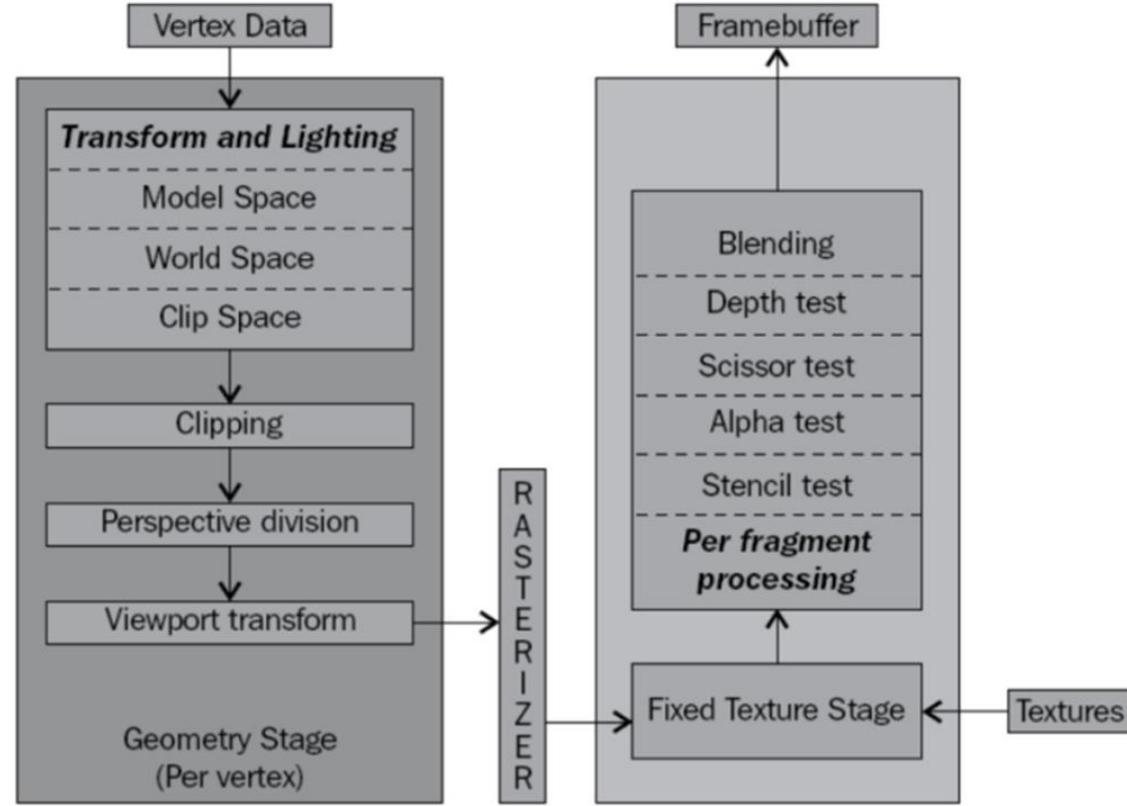
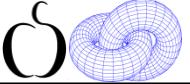
- Общие параметры построения многоугольников:

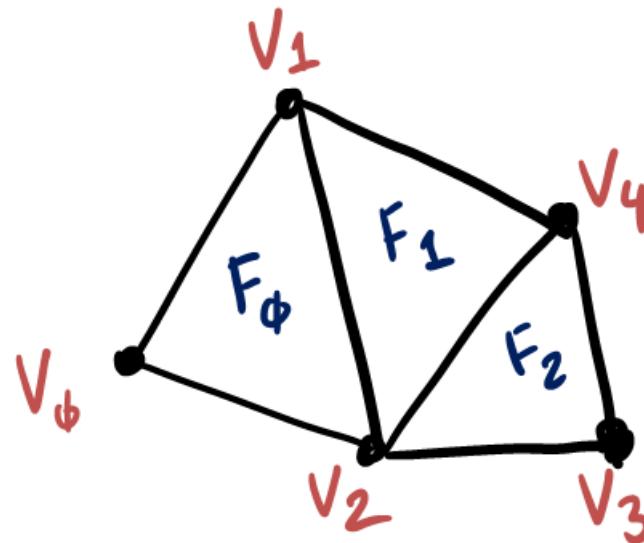
`glPolygonMode(кого, как);`

кого — `GL_BACK` или `GL_FRONT` или `GL_FRONT_AND_BACK`

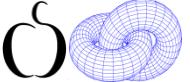
как — `GL_FILL` или `GL_LINE` или `GL_POINT`







$$V = \{V_0, V_1, V_2, V_3, V_4\}$$
$$I = \{0, 2, 1, 2, 4, 1, 2, 3, 4\}$$



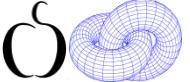
VBO – vertex buffer object (примитив в видеопамяти):

для работы необходимы - массив вершин и буфер вершин:  
vertex array vertex buffer  
(описание данных) (сами данные)  
кто где VERTEX

массив вершин - *VertexArray* –  
связка посылаемых данных сверху вниз (*Layout*).

буфер вершин - *VertexBuffer* –  
массив данных, отсылаемых в видеокарту.





# Vertex Buffer Object (VBO)

Хранение вершин:

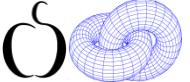
```
/* Структура хранения данных о вершине */
typedef struct tagvg4VERTEX
{
    VEC P;      /* позиция */
    VEC2 T;     /* текстурные координаты */
    VEC N;      /* нормаль */
    VEC4 C;     /* Цвет */
} vg4VERTEX;

typedef struct tagVEC2
{
    float X, Y;    /* координаты */
} VEC2;

typedef struct tagVEC
{
    float X, Y, Z;  /* координаты */
} VEC;

typedef struct tagVEC4
{
    float X, Y, Z, W; /* координаты */
} VEC4;
```





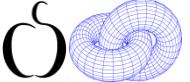
Инициализационный этап

```
int VA, VBuf;  
  
glGenBuffers(1, &VBuf);  
glGenVertexArrays(1, &VA);  
  
/* делаем активным массив вершин */  
glBindVertexArray(VA);
```

заносим данные в буфер вершин:

```
/* делаем активным буфер */  
glBindBuffer(GL_ARRAY_BUFFER, VBuf);  
/* передаем данные (NumOfV - количество вершин, V - массив вершин) */  
glBufferData(GL_ARRAY_BUFFER, sizeof(vg4VERTEX) * NumOfV, V, GL_STATIC_DRAW);
```





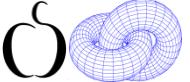
# Vertex Buffer Object (VBO)

указываем в массиве вершин буфер и какие данные содержит:

```
/* присоединяем к массиву вершин буфер с данными (если еще не делали) */
glBindBuffer(GL_ARRAY_BUFFER, VBuf);

/* задаем порядок данных */
/* Layout (номер атрибута),
 * количество компонент,
 * mun,
 * надо ли нормировать,
 * размер в байтах одного элемента буфера (stride),
 * смещение в байтах до начала данных */

glVertexAttribPointer(0, 3, GL_FLOAT, FALSE, sizeof(vg4VERTEX),
                      (VOID *)0); /* позиция */
glVertexAttribPointer(1, 2, GL_FLOAT, FALSE, sizeof(vg4VERTEX),
                      (VOID *)sizeof(VEC)); /* текстурные координаты */
glVertexAttribPointer(2, 3, GL_FLOAT, FALSE, sizeof(vg4VERTEX),
                      (VOID *)(sizeof(VEC) + sizeof(VEC2))); /* нормаль */
glVertexAttribPointer(3, 4, GL_FLOAT, FALSE, sizeof(vg4VERTEX),
                      (VOID *)sizeof(VEC) * 2 + sizeof(VEC2)); /* цвет */
```



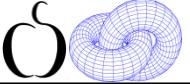
# Vertex Buffer Object (VBO)

```
/* включаем нужные аттрибуты (Layout) */
glEnableVertexAttribArray(0);
glEnableVertexAttribArray(1);
glEnableVertexAttribArray(2);
glEnableVertexAttribArray(3);

/* выключили массив вершин */
glBindVertexArray(0);
```

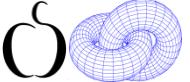
удаление

```
/* делаем активным массив вершин */
glBindVertexArray(VA);
/* "отцепляем" буфер */
glBindBuffer(GL_ARRAY_BUFFER, 0);
glDeleteBuffers(1, &VBuf);
/* делаем неактивным массив вершин */
glBindVertexArray(0);
glDeleteVertexArrays(1, &VA);
```



Отрисовка:

```
/* делаем активным массив вершин */
glBindVertexArray(VA);
/* отрисовка */
glDrawArrays(GL_TRIANGLES, 0, NumOfV);
/* выключили массив вершин */
glBindVertexArray(0);
```



# Vertex Buffer Object (VBO)

Индексы:

инициализационный этап

```
int IBuf;  
...  
glGenBuffers(1, &IBuf);
```

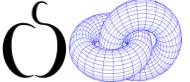
заносим данные в буфер индексов:

```
/* делаем активным буфер */  
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, IBuf);  
/* передаем данные (NumOfI - количество индексов, I - массив индексов) */  
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(INT) * NumOfI, I, GL_STATIC_DRAW);
```

удаление

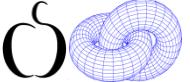
```
glDeleteBuffers(1, &IBuf);
```



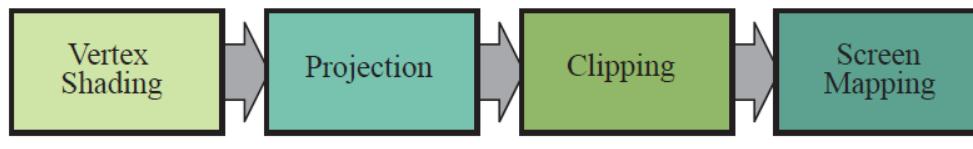
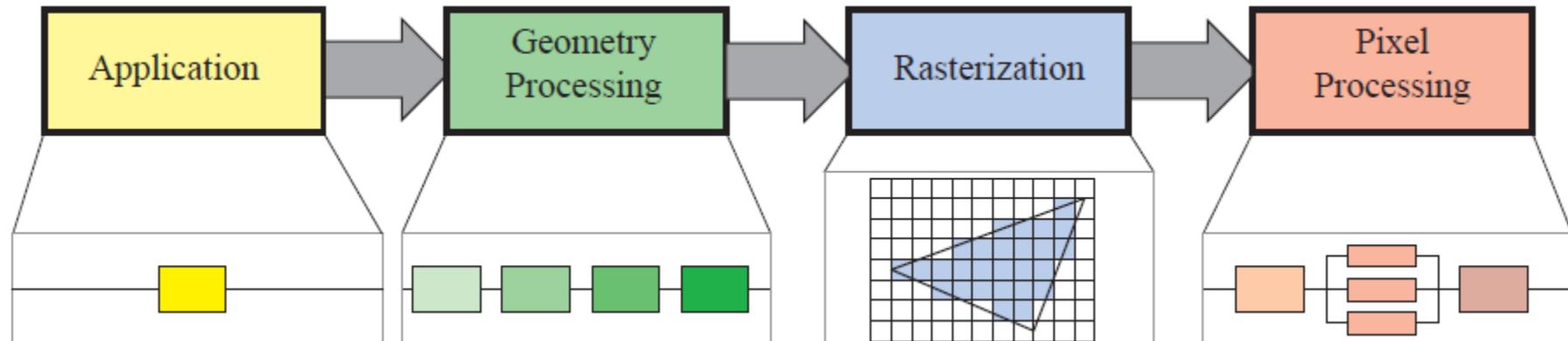


Отрисовка (индексированные примитивы):

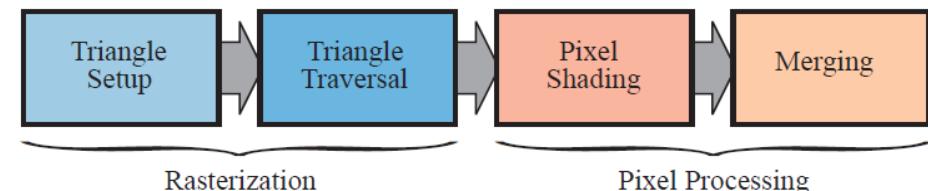
```
/* делаем активным массив вершин */
glBindVertexArray(VA);
/* делаем активным массив индексов */
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, IBuf);
/* отрисовка */
glDrawElements(GL_TRIANGLES, NumOfI, GL_UNSIGNED_INT, NULL);
/* выключили индексы */
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
/* выключили массив вершин */
glBindVertexArray(0);
```



# Графический конвейер

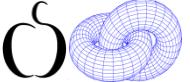


Geometry Processing



Rasterization

Pixel Processing



*Vertex data ->*

**Vertex shader ->**

**Tessellation control shader ->**

**Tessellation evaluation shader ->**

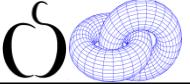
**Geometry shader ->**

*Rasterizer (assembling primitives, rasterization) ->*

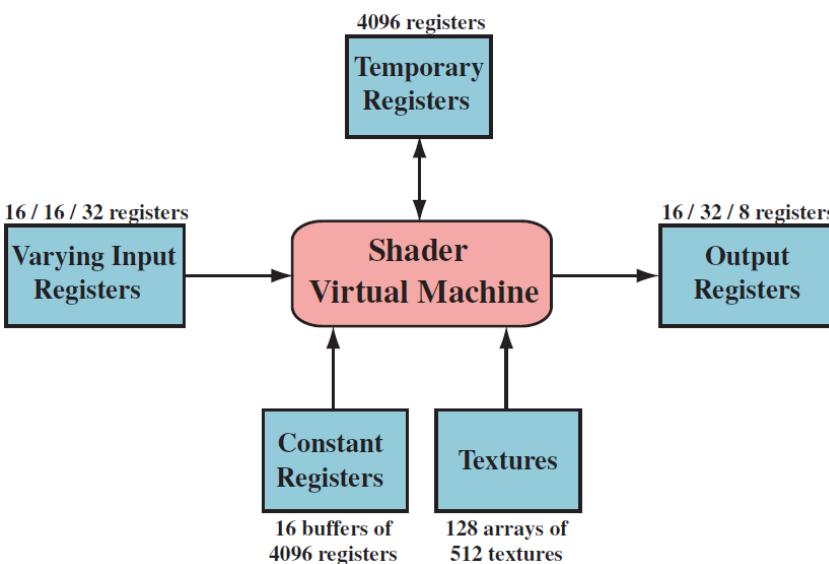
**Fragment shader ->**

*Raster operation (stencil, alpha, scissor, depth, blending) ->*

*Framebuffer*



**Cg (Nvidia, 2004)**  
**GLSL (OpenGL)**  
**HLSL (Direct3D)**



**CPU -> Vertex attributes -> VS**

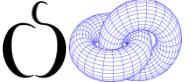
-- передаем для каждой вершины набор атрибутов (позиция, цвет и т.п.) через VBO

**shader -> Varying variables -> shader**

-- переменные от каждой вершины могут быть переданы для дальнейшей обработки, например, после вершинного шейдера они интерполируются от вершины к вершине во время растеризации и строятся для каждого пикселя и приходят во фрагментный шейдер, от фрагментного шейдера выходные данные поступают в буфер кадра

**Uniform variables - globals**

-- глобальные переменные, доступные во всех шейдерах одной шейдерной программы (например, матрица преобразований, время, текстурные сэмплеры, параметры освещения и т.п.)



Базовые типы:

`bool` - true | false

`int`, `uint`

`sampler` (`sampler1D`, `sampler 2D`, `sampler3D`)

`float`

структурные:

Vectors:

`bvec2`, `bvec3`, `bvec4` (bool)

`ivec2`, `ivec3`, `ivec4` (int)

`uvec2`, `uvec3`, `uvec4` (uint)

`vec2`, `vec3`, `vec4` (float)

`dvec2`, `dvec3`, `dvec4` (double)

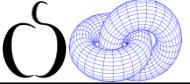
Matrices:

`mat2`, `mat3`, `mat4` (2x2, 3x3, 4x4)

`mat2x3`, `mat2x4`, `mat3x2`, `mat3x4`, `mat4x2`, `mat4x3` (rows x columns)

`dmat2`, `dmat3`, `dmat4`

`dmat2x3`, `dmat2x4`, `dmat3x2`, `dmat3x4`, `dmat4x2`, `dmat4x3`



Запись числовых констант:

`1.5` - *float*

`1.5f` - *float*

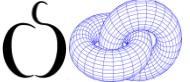
`1.5lf` - *double*

Инициализация:

```
float a = 13.47;  
bool is_space = false;  
ivec3 a = ivec3(1, 2, 3);  
uvec3 b = uvec3(1, -2, 3);  
vec3 v = vec3(1.0, 2.0, 3);  
vec4 v1 = vec4(1.0, vec3(4, 5, 6));  
vec4 v2 = vec4(vec3(4, 5, 6), 8);  
vec4 v3 = vec4(vec2(4, 5), vec2(13, 8));  
vec4 v3 = vec4(0);      <=>      vec4(0, 0, 0, 0);
```

заполнение матрицы по **столбцам**:

```
mat3 m = mat3(vec3(1, 2, 3),  
               vec3(3, 4, 5),  
               vec3(6, 7, 8));
```



# Основы GLSL

Доступ к компонентам векторов:

```
vec2 p;
p[0] = 1.0;    <-- x
p.x = 1;
p.y = 3.3;
```

любой вектор - это набор полей:

```
{x, y, z, w} - геометрия
{r, g, b, a} - цвет
{s, t, p, q} - текстурирование
p.x == p.r == p.s
```

*swizzle*:

```
vec3 v = p.xxy;  <=> vec3(p.x, p.x, p.y);
vec4 c = v.rrrr;
vec4 r = v.rryy;
```

Доступ к компонентам матриц:

```
mat4 m;
m[0] = vec4(1); -- весь первый столбец в 1
m[2][1] = 30.50;
vec4 v3 = vec4(vec2(4, 5), vec2(13, 8));
```

Операции:

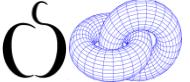
```
mat3 T, R, M;
vec3 v, b;
float f;
b = v + f;
(b.x = v.x + f, b.y = v.y + f, ... )
```

!!! используется нотация вектор-столбец:  
 $b = M * v;$   
 $M = R * T;$   
 $b = R * T * v;$

преобразования скалярных типов:

```
int a = int(47.13);
bool c = false;
float x = float(c); <-- 0.0
```





# Основы GLSL

Агрегатные типы:

*структуры:*

```
struct Light
{
    vec3 Pos;
    vec4 Color;
    float Attenuation;
};
```

```
Light L = Light(vec3(8, 8, 8), vec4(1, 0, 0, 1), 0.47);
```

*массивы:*

```
Light ls[10];
for (int i = 0; i < 10; i++)
{
    ls[i].Attenuation = i / 10.0;
}
for (int i = 0; i < ls.length(); i++)
    . .
vec4 v;
v.length() --> 4
```

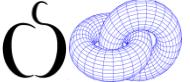
Препроцессор:

```
#error
#pragma
#version
#define
#if #ifdef #ifndef #else #endif #elif
```

Точка входа:

```
void main( void )
{
}
```





Дополнительные функции:

```
float t = dot(V1, V2);
vec3 V3 = cross(V1, V2);

T X = max(X1, X2)
T X = min(X1, X2)

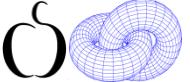
T X = clamp(X1, A, B) <=> min(max(X1, A), B) <=> (X1 < A ? A : X1 > B : B : X)
T X = mix(A, B, t)      <=> A * (1 - t) + B * t

sin cos tan asin acos atan(x) atan(y, x) pow(x, y) log
exp sqrt inversesqrt abs sign floor round trunc ceil mod

float t = length(V1)
float t = distance(P1, P2)
vec V = normalize(V1)

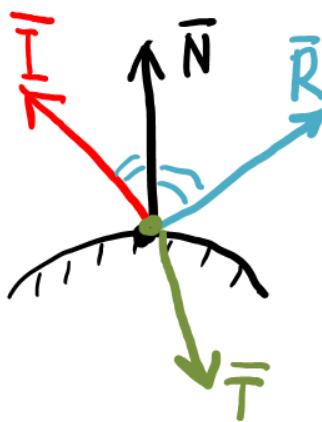
mat M;
mat T = inverse(M)
mat T = transpose(M)
float d = determinant(M)
```



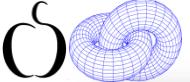


Функции для моделирования освещения:

```
vec V = faceforward(N, I, Nref) <=> dot(I, Nref) < 0 ? N : -N
vec R = reflect(I, N) <=> I - 2 * dot(N, I) * N
vec T = refract(I, N, eta) <=> k = 1.0 - eta * eta * (1.0 - dot(N, I) * dot(N, I));
                                         if (k < 0.0)
                                         V = vec(0.0);
else
    V = eta * I - (eta * dot(N, I) + sqrt(k)) * N;
```



Справочная таблица: <https://www.khronos.org/files/opengl46-quick-reference-card.pdf>



# Загрузка шейдера

```
int res, NumOfShaders,
    shd_type[] =
{
    GL_VERTEX_SHADER, GL_TESS_CONTROL_SHADER,
    GL_TESS_EVALUATION_SHADER, GL_GEOMETRY_SHADER,
    GL_FRAGMENT_SHADER
};
unsigned prg = 0, shd[5];
char *txt, buf[1000];

/* Load text file */
txt = LoadTextFile("a.vert");
/* Create shader */
shd[0] = glCreateShader(shd_type[0]);
/* Attach text to shader */
glShaderSource(shd[0], 1, &txt, NULL);
free(txt);
/* Compile shader */
glCompileShader(shd[i]);
glGetShaderiv(shd[i], GL_COMPILE_STATUS, &res);
if (res != 1)
{
    glGetShaderInfoLog(shd[0], sizeof(buf), &res, buf);
    ErrorLog(buf);
    return;
}
. . .
```

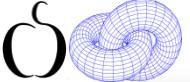
```
/* Create program */
prg = glCreateProgram();
/* Attach shaders to program */
for (i = 0; i < NumOfShaders; i++)
    glAttachShader(prg, shd[i]);
/* Link program */
glLinkProgram(prg);
glGetProgramiv(prg, GL_LINK_STATUS, &res);
if (res != 1)
{
    glGetProgramInfoLog(prg, sizeof(buf), &res, buf);
    ErrorLog(buf);
    return;
}
return prg;
```

Удаление шейдера:

```
unsigned i, n, shds[5];

glGetAttachedShaders(Prg, 5, &n, shds);
for (i = 0; i < n; i++)
{
    glDetachShader(Prg, shds[i]);
    glDeleteShader(shds[i]);
}
glDeleteProgram(Prg);
```





# Простейший шейдер

*vert.glsl*

```
// версия языка шейдера (4.5)
#version 450

// кто куда приходит
Layout(Location = 0) in vec3 InPosition;
Layout(Location = 1) in vec2 InTexCoord;
Layout(Location = 2) in vec3 InNormal;
Layout(Location = 3) in vec4 InColor;

// глобальные переменные (произведение
// матриц: World * View * Proj)
uniform mat4 MatrWVP;

// выходные параметры (varying)
out vec4 DrawColor;

void main( void )
{
    gl_Position = MatrWVP * vec4(InPosition, 1);
    DrawColor = InColor;
}
```

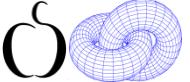
*frag.glsl*

```
// версия языка шейдера (4.5)
#version 450

// выходные параметры - цвет рисования
Layout(Location = 0) out vec4 OutColor;
// входные параметры (varying)
in vec4 DrawColor;

void main( void )
{
    OutColor = DrawColor;
}
```





```
int loc;

glUseProgram(PrgId);

if ((loc = glGetUniformLocation(PrgId, "Matrix")) != -1)
    glUniformMatrix4fv(loc, 1, FALSE, указатель на первый элемент матрицы);

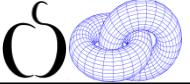
if ((loc = glGetUniformLocation(PrgId, "Vector")) != -1)
    glUniform3fv(loc, 1, указатель на первую координату вектора);

if ((loc = glGetUniformLocation(PrgId, "FloatNumber")) != -1)
    glUniform1f(loc, вещественное число);

if ((loc = glGetUniformLocation(PrgId, "IntegerNumber")) != -1)
    glUniform1i(loc, целое число);

. . . отрисовка . . .

glUseProgram(0);
```



# Текстурирование

Генерация «свободных» номеров текстур в массив:

```
int TexNames[4];  
glGenTextures(4, TexNames);
```

Переключение текстур (установка номера активной текстуры):

```
glBindTexture(GL_TEXTURE_2D, number);
```

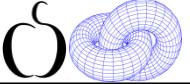
Функции задания изображений в текстуру:

```
glTexImage2D(GL_TEXTURE_2D,  
    0,                      уровень MIP (multum in parvo) 0 - базовая картинка  
    1,                      количество компонент на точку (1, 2, 3, 4)  
    w, h,                  ширина и высота (должны быть степенью 2)  
    0,                      наличие границы  
    GL_LUMINANCE,          трактовка компонент точки (GL_LUMINANCE, GL_BGR_EXT, GL_BGRA_EXT)  
    GL_UNSIGNED_BYTE,       тип каждой компоненты (GL_UNSIGNED_BYTE, GL_FLOAT)  
    ptr);                  указатель на массив с компонентами
```

Автоматическая генерация текстур всех размеров «сниз», начиная с указанного размера

```
gluBuild2DMipmaps(GL_TEXTURE_2D,  
    3,                      количество компонент на точку (1, 2, 3, 4)  
    w, h,                  ширина и высота - любые  
    GL_BGR_EXT,            трактовка компонент точки (GL_LUMINANCE, GL_BGR_EXT, GL_BGRA_EXT)  
    GL_UNSIGNED_BYTE,       тип каждой компоненты (GL_UNSIGNED_BYTE, GL_FLOAT)  
    ptr);                  указатель на массив с компонентами
```





# Текстурирование

*Управление фильтрацией:*

```
glTexParameteri(GL_TEXTURE_2D,  
    GL_TEXTURE_MIN_FILTER, GL_NEAREST);  
    GL_LINEAR  
    GL_NEAREST_MIPMAP_NEAREST  
    GL_LINEAR_MIPMAP_NEAREST  
    GL_NEAREST_MIPMAP_LINEAR  
    GL_LINEAR_MIPMAP_LINEAR  
  
glTexParameteri(GL_TEXTURE_2D,  
    GL_TEXTURE_MAG_FILTER, GL_NEAREST);  
    GL_LINEAR
```

*Параметры наложения (постор текстуры по направлениям):*

```
glTexParameteri(GL_TEXTURE_2D,  
    GL_TEXTURE_WRAP_S, GL_REPEAT);  
    GL_TEXTURE_WRAP_T GL_CLAMP
```

*Параметры выравнивания - на какое число должна делится нацело длина одной строки в байтах:*

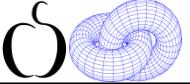
*для установки изображения в текстуру (из программы в OpenGL)*

```
glPixelStorei(GL_UNPACK_ALIGNMENT, число(1,2,4,...));
```

*для чтения изображения (от OpenGL в программу)*

```
glPixelStorei(GL_PACK_ALIGNMENT, число(1,2,4,...));
```





## Функции работы с новыми форматами:

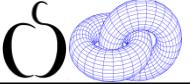
```
glTexStorage2D(GL_TEXTURE_2D,
```

- количество уровней MipMap
- формат RGB float (GL\_RGBA32UI, GL\_R32F, ...)
- размер текстуры (w, h)

```
glTexSubImage2D(GL_TEXTURE_2D,
```

- уровень MipMap
- смещение изображения в текстуре (dx, dy)
- размер вставляемого изображения в текстуру (w, h)
- формат RGB – три компоненты
- тип компоненты
- сам массив





# Текстурирование

Выборка цвета из текстуры осуществляется сэмплерами (*samplers*)

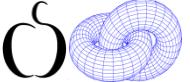
```
glActiveTexture(GL_TEXTURE0 + SamplerNo);    -- активация сэмплера  
glBindTexture(GL_TEXTURE_2D, TexId);           -- подключение в него текстуры
```

На шейдере (фрагментном):

```
// текстурные координаты от вершинного шейдера  
in vec2 DrawTexCoord;  
  
// указания какой сэмплер как называется  
layout(binding = 0) uniform sampler2D TextureKd;  
layout(binding = 1) uniform sampler2D TextureMask;  
  
void main( void )  
{  
    . . .  
    // получение цвета  
    vec4 tex_color = texture(TextureKd, DrawTexCoord);  
    . . .  
}
```

texture ? texelFetch





*Vertex data* ->

**Vertex shader** ->

Tessellation control shader ->

Tessellation evaluation shader ->

**Geometry shader** ->

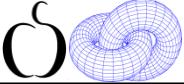
*Rasterizer (assembling primitives, rasterization)* ->

**Fragment shader** ->

*Raster operation (stencil, alpha, scissor, depth, blending)* ->

*Framebuffer*

**Vertex shader** -> "gl\_Position" -> **Geometry shader** -> {"gl\_Position"}+**primitive\_type** ->...



# Геометрический шейдер

На вход - примитивы:

`points (GL_POINTS)`

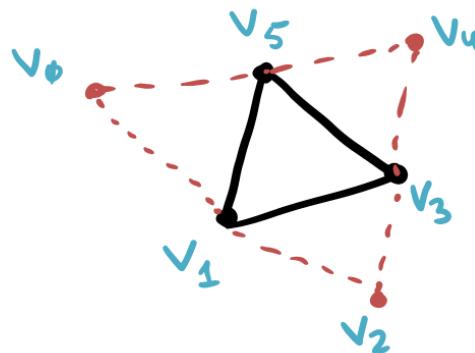
`lines (GL_LINES, GL_LINE_STRIP, GL_LINES_LOOP)`

`lines_adjacency (GL_LINES_ADJACENCY, GL_LINE_STRIP_ADJACENCY)`



`triangles (GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN)`

`triangles_adjacency (GL_TRIANGLES_ADJACENCY, GL_TRIANGLE_STRIP_ADJACENCY)`



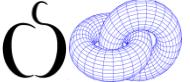
На выход - примитивы:

`points`

`line_strip`

`triangle_strip`





Структура описания 1-й вершины (в глобальном массиве `gl_in`):

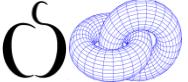
```
in gl_PerVertex
{
    vec4 gl_Position;    -- с вершинного (тесселяционного) шейдера
    ...
} gl_in[];
```

на выходе: `out vec4 gl_Position;`

порождение вершины: `EmitVertex();`

окончание примитива: `EndPrimitive();`





# Геометрический шейдер: пример - частицы

Геометрический шейдер

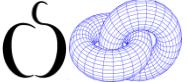
```
#version 430

layout(points) in;
layout(triangle_strip, max_vertices = 40) out;

uniform mat4 MatrWVP;
uniform float Time;
uniform vec3 CamRight;
uniform vec3 CamUp;
out vec2 FDrawTexCoord;
void main(_void )
{
    int n = 10;
    float t = Time;
    float s = 3.24;
    vec3 p = gl_in[0].gl_Position.xyz;
    for (int i = 0; i < n; i++)
    {
        float angle = Time * i;
        float si = sin(angle), co = cos(angle);
        vec3 r = CamRight * co + CamUp * si;
        vec3 u = CamRight * -si + CamUp * co;
```

```
    p += vec3(0, s / 2, 0);
    gl_Position = MatrWVP * vec4(p + r * -s + u * s, 1);
    FDrawTexCoord = vec2(0, 1);
    EmitVertex();
    gl_Position = MatrWVP * vec4(p + r * -s + u * -s, 1);
    FDrawTexCoord = vec2(0, 0);
    EmitVertex();
    gl_Position = MatrWVP * vec4(p + r * s + u * s, 1);
    FDrawTexCoord = vec2(1, 1);
    EmitVertex();
    gl_Position = MatrWVP * vec4(p + r * s + u * -s, 1);
    FDrawTexCoord = vec2(1, 0);
    EmitVertex();
    EndPrimitive();
}
```





# Геометрический шейдер: пример - частицы

Вершинный шейдер

```
#version 430

layout(location = 0) in vec3 InPosition;

void main( void )
{
    gl_Position = vec4(InPosition, 1);
}
```

Фрагментный шейдер

```
#version 430

layout(location = 0) out vec4 OutColor;

uniform float Time;
uniform bool IsWireframe;
layout(binding = 0) uniform sampler2D Texture0;

in vec2 FDrawTexCoord;

void main( void )
{
    vec4 c = texture2D(Texture0, FDrawTexCoord);

    if (IsWireframe)
        c = vec4(1, 1, 0.5, 1);
    OutColor = c;
}
```



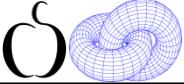
COMPUTER  
SCIENCE  
CLUB



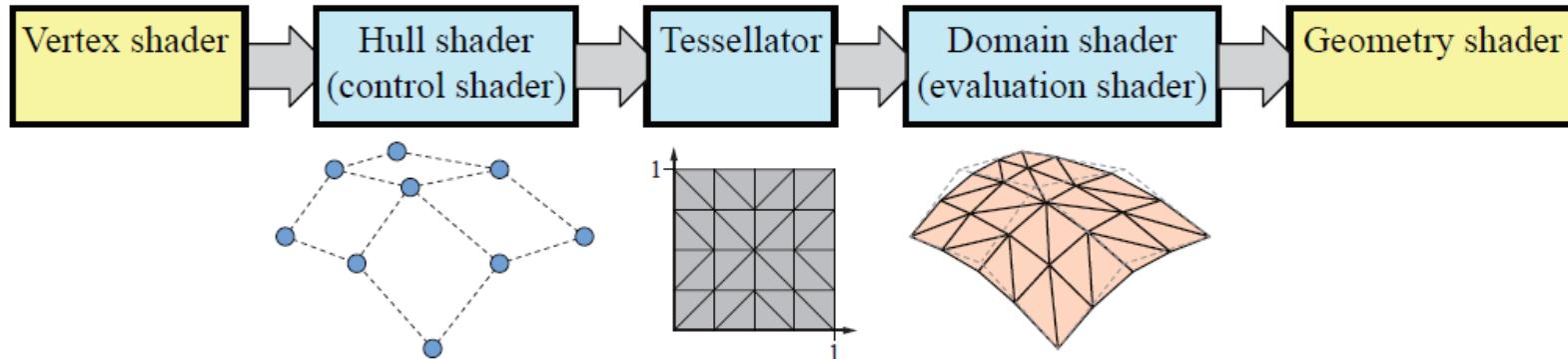
Computer Graphics Support Group  
Санкт-Петербургский губернаторский  
ФИЗИКО-МАТЕМАТИЧЕСКИЙ ЛИЦЕЙ № 30



Галинский В.А.  
Графика реального времени. OpenGL



# Тесселяционный шейдер



*Vertex data ->*

*Vertex shader ->*

*Tessellation control shader ->*

*Tessellation evaluation shader ->*

*Geometry shader ->*

*Rasterizer (assembling primitives, rasterization) ->*

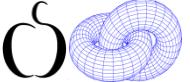
*Fragment shader ->*

*Raster operation (stencil, alpha, scissor, depth, blending) ->*  
*Framebuffer*

**Input: \*** `(GL_PATCHES)`

**Output:** `layout(isolines) out;`  
`layout(triangles) out;`  
`layout(quads) out;`





Тип примитива OpenGL:

**GL\_PATCHES**

Указываем количество вершин в наборе:

```
glPatchParameteri(GL_PATCH_VERTICES, N);
```

При выводе:

• • •

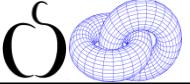
```
glDrawElements(GL_PATCHES, NumberOfIndices, GL_UNSIGNED_INT, NULL);
```

или

```
glDrawArrays(GL_PATCHES, 0, NumberOfVertices);
```

• • •



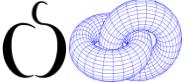


## *Vertex shader (\*.VERT):*

```
#version 430

layout(location = 0) in vec3 InPosition;

void main( void )
{
    gl_Position = vec4(InPosition, 1);
}
```



*Tessellation control shader (\*.CTRL):*

```
#version 430
```

```
layout(vertices = 4) out; // Количество вершин в одном наборе
```

```
void main( void )
```

```
{
```

```
/* Контрольный шейдер вызывается для каждой вершины набора (у нас в нем 4-е вершины
 * смотри Layout. Порядковый номер вершины находится в номере gl_InvocationID
 * gl_in и gl_out - массивы со входными и выходными данными по всем вершинам набора
 * как и в геометрическом шейдере. Мы просто копируем вершину дальше */
```

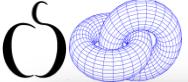
```
gl_out[gl_InvocationID].gl_Position = gl_in[gl_InvocationID].gl_Position;
```

```
gl_TessLevelOuter[0] = 1; // Указываем количество изолиний
```

```
gl_TessLevelOuter[1] = 47; // Указываем величину разбиения ломаной
```

```
}
```





# Тесселяционный шейдер: пример - кривая Бэзье

Tesselation evaluation shader:

```
#version 430

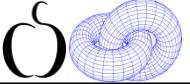
/* Указываем тип построения разбиения - ломаная (изолиния) */
layout(isolines) in;

uniform mat4 MatrWVP;

vec3 Bezier( vec3 P0, vec3 P1, vec3 P2, vec3 P3, float t )
{
    return P0 * (1 - t) * (1 - t) * (1 - t) +
           P1 * 3 * (1 - t) * (1 - t) * t +
           P2 * 3 * (1 - t) * t * t +
           P3 * t * t * t;
}
```

```
void main( void )
{
    /* Получаем по 4 точки из массива gl_in. Переменная gl_TessCoord в координатах
     * содержит интерполяционный параметр. У нас линия, одно измерение используем только
     * gl_TessCoord.x. В случае бикубических поверхностей - использовались бы
     * gl_TessCoord.x и gl_TessCoord.y. Каждый параметр пробегает от 0 до 1 с шагом,
     * заданным разбиением в контролльном шейдере. Вычислительный шейдер вызывается для
     * каждой точки тесселяции (подразбиения) персонально. */
    gl_Position = MatrWVP *
        vec4(Bezier(gl_in[1].gl_Position.xyz,
                    gl_in[1].gl_Position.xyz +
                    (gl_in[2].gl_Position.xyz - gl_in[0].gl_Position.xyz) / 6,
                    gl_in[2].gl_Position.xyz +
                    (gl_in[1].gl_Position.xyz - gl_in[3].gl_Position.xyz) / 6,
                    gl_in[2].gl_Position.xyz,
                    gl_TessCoord.x), 1);
}
```





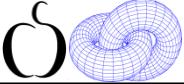
*Fragment shader:*

```
#version 430

layout(location = 0) out vec4 OutColor;

void main( void )
{
    OutColor = vec4(1, 0, 1, 1); // Рисуем малиновую точку
}
```





# Тесселяционный шейдер: пример - поверхность

Tessellation control shader (\*.CTRL):

```
#version 430

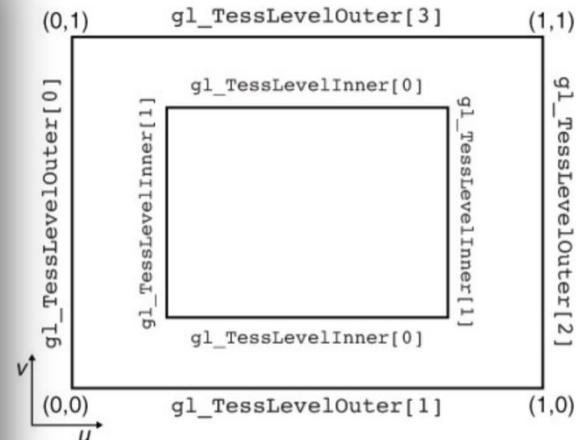
layout(vertices = 16) out;

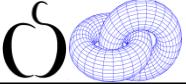
uniform int Addon0;
uniform int Addon1;
uniform int Addon2;

void main( void )
{
    gl_out[gl_InvocationID].gl_Position = gl_in[gl_InvocationID].gl_Position;

    // Outer split values
    gl_TessLevelOuter[0] = Addon0;    // by U0
    gl_TessLevelOuter[1] = Addon0;    // by V0
    gl_TessLevelOuter[2] = Addon0;    // by U1
    gl_TessLevelOuter[3] = Addon0;    // by V1

    // Inner split values
    gl_TessLevelInner[0] = Addon1;   // by U
    gl_TessLevelInner[1] = Addon2;   // by V
}
```





# Тесселяционный шейдер: пример - поверхность

*Tesselation evaluation shader: Tessellation control shader (\*.CTRL):*

```
#version 430

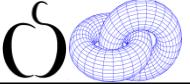
layout(quads) in;

uniform mat4 MatrWVP;

void main( void )
{
    float u = gl_TessCoord.x, v = gl_TessCoord.y;

    vec3 p = ...
        точка на поверхности
        опорные точки (4x4) - gl_in[i * 4 + j].gl_Position.xyz
        параметры - u, v

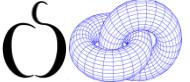
    gl_Position = MatrWVP * vec4(p, 1);
}
```



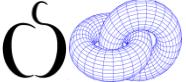
## Frame Buffer Object (FBO) – Render Target

- создать дескриптор FBO
- создать текстуру для FBO (или несколько текстур) и закрепить за FBO
- создать буфер глубины и закрепить за FBO





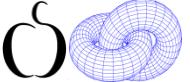
```
/* FBO depended data */
unsigned
    VG4_RndFBO,          /* Frame buffer object descriptor */
    VG4_RndRenderTex,    /* Render target texture */
    VG4_RndDepthBuf;     /* Render target buffer */
```



# FBO – Frame Buffer Object

```
RndTargetsInit:  
/* Fragment shader texture attachment list */  
UINT DrawBuffers[] = {GL_COLOR_ATTACHMENT0};  
  
/* Create and bind FBO */  
glGenFramebuffers(1, &VG4_RndFBO);  
 glBindFramebuffer(GL_FRAMEBUFFER, VG4_RndFBO);  
  
/* Create and bind render texture */  
glGenTextures(1, &VG4_RndRenderTex);  
 glBindTexture(GL_TEXTURE0);  
 glBindTexture(GL_TEXTURE_2D, VG4_RndRenderTex);  
 glTexStorage2D(GL_TEXTURE_2D, 1, GL_RGBA8, VG4_RndFrameW, VG4_RndFrameH);  
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);  
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);  
  
/* Link render texture to FBO */  
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, VG4_RndRenderTex, 0);  
...
```





# FBO – Frame Buffer Object

```
...
/* Create and bind depth buffer */
glGenRenderbuffers(1, &VG4_RndDepthBuf);
 glBindRenderbuffer(GL_RENDERBUFFER, VG4_RndDepthBuf);
 glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH_COMPONENT, VG4_RndFrameW, VG4_RndFrameH);

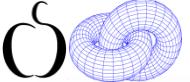
/* Link depth buffer to FBO */
glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_RENDERBUFFER, VG4_RndDepthBuf);

/* Set fragment shader texture attachment list:
 * array contents layout slot redirection for every output value */
glDrawBuffers(1, DrawBuffers);

if (glCheckFramebufferStatus(GL_FRAMEBUFFER) != GL_FRAMEBUFFER_COMPLETE)
    return;

/* Go back to default render buffer */
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```



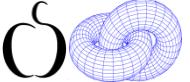


# FBO – Frame Buffer Object

RndTargetsClose:

```
/* Delete FBO primitive */
VG4_RndPrimFree(&VG4_RndFBOFramePrim);
/* Bind FBO */
glBindFramebuffer(GL_FRAMEBUFFER, VG4_RndFBO);
/* Unlink render texture from FBO */
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, 0, 0);
/* Delete render texture */
glDeleteTextures(1, &VG4_RndRenderTex);
/* Unlink depth buffer from FBO */
glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_RENDERBUFFER, 0);
/* Delete depth buffer */
glDeleteRenderbuffers(1, &VG4_RndDepthBuf);
/* Unlink FBO */
glBindFramebuffer(GL_FRAMEBUFFER, 0);
/* Delete FBO */
glDeleteFramebuffers(1, &VG4_RndFBO);
```

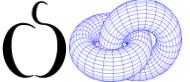




# FBO – Frame Buffer Object

```
RndTargetsStart: // вызываем из RndStart()
/* Bind FBO */
glBindFramebuffer(GL_FRAMEBUFFER, VG4_RndFBO);

/* Set viewport */
glViewport(0, 0, VG4_RndFrameW, VG4_RndFrameH);
glClearColor(0.3, 0.5, 0.7, 1);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```



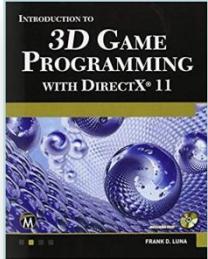
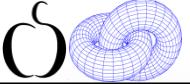
# FBO – Frame Buffer Object

```
RndTargetsEnd: // вызываем из RndEnd()
    /* Set default FBO */
    glBindFramebuffer(GL_FRAMEBUFFER, 0);
    glViewport(0, 0, VG4_RndFrameW, VG4_RndFrameH);

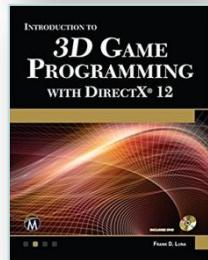
    /* Draw post frame primitive (full screen rectangle) to screen */
    glUseProgram(шейдер обработки FBO);

    /* Send render target */
    glActiveTexture(GL_TEXTURE3); // в примере используется 3
    glBindTexture(GL_TEXTURE_2D, VG4_RndRenderTex);

    /* Draw frame */
    glPushAttrib(GL_ALL_ATTRIB_BITS); // запоминаем состояние атрибутов
    glDisable(GL_DEPTH_TEST);
    glDisable(GL_BLEND);
    glDisable(GL_CULL_FACE);
    ... выводим полноэкранный примитив FBO ...
    glPopAttrib(); // восстанавливаем состояние атрибутов
    glUseProgram(0);
```



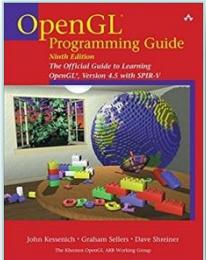
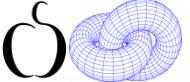
Frank Luna, «*Introduction to 3D Game Programming with DirectX 11*», Mercury Learning & Information, 2012.



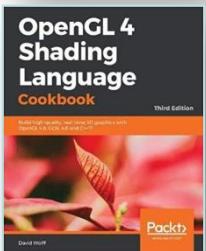
Frank Luna, «*Introduction to 3D Game Programming with DirectX 12*», Mercury Learning & Information, 2016.



Грэхем Селлерс, «*Vulkan. Руководство разработчика*», ДМК Пресс, 2017.



John Kessenich, Graham Sellers, Dave Shreiner,  
**«OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.5 with SPIR-V (9th Edition)»**,  
Addison-Wesley Professional, 2016.

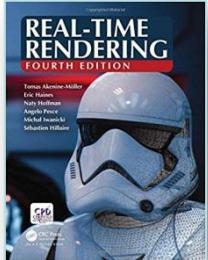
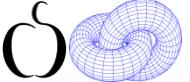


David Wolff, **«OpenGL 4 Shading Language Cookbook: Build high-quality, real-time 3D graphics with OpenGL 4.6, GLSL 4.6 and C++17, 3rd Edition»**,  
Packt Publishing, 2018.

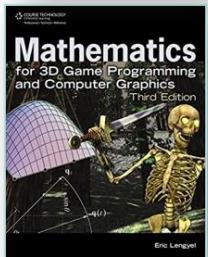


Дэвид Вольф, **«OpenGL 4. Язык шейдеров. Книга рецептов»**,  
ДМК Пресс, 2015.





Tomas Akenine-Möller, Eric Haines, Naty Hoffman,  
**«Real-Time Rendering, Fourth Edition»**,  
A K Peters/CRC Press, 2018.

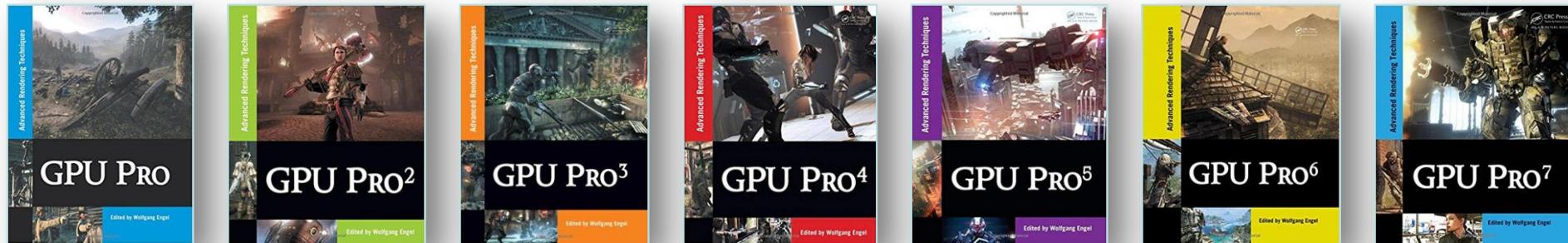
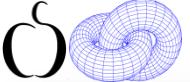


Eric Lengyel,  
**«Mathematics for 3D Game Programming and Computer Graphics, Third Edition 3rd Edition»**,  
Cengage Learning PTR, 2011



Elmar Eisemann, Michael Schwarz, Ulf Assarsson, Michael Wimmer,  
**«Real-Time Shadows»**,  
A K Peters/CRC Press, 2011.





Wolfgang Engel (Editor), «**GPU Pro: Advanced Rendering Techniques**», A K Peters/CRC Press, 2010.

«**GPU Pro2**», 2011; «**GPU Pro3**», 2012; «**GPU Pro4**», 2013; «**GPU Pro5**», 2014; «**GPU Pro6**», 2015; «**GPU Pro7**», 2016.



Wolfgang Engel, «**GPU Zen: Advanced Rendering Techniques**», Bowker Identifier Services, 2017